# DEFINIÇÃO DE METODOLOGIA DE APLICAÇÃO DE COMUNICAÇÃO ASSÍNCRONA DE FUNÇÕES NA ARQUITETURA CUDA

**Cássio Silva de Sá Santos[1] e Angelo Amâncio Duarte[2];**

1. Bolsista PIBIC/FAPESB, Graduando em Engenharia de Computação, Universidade Estadual de Feira de Santana, e-mail: sss.cassio@gmail.com

2. Orientador, Departamento de Tecnologia, Universidade Estadual de Feira de Santana, e-mail: angeloduarte@uefs.br

**PALAVRAS-CHAVE:** Computação de Alto Desempenho; Cuda; GPU.

## INTRODUÇÃO

A Computação de Alto Desempenho, ou High Performance Computing (HPC) em inglês, é o conjunto de técnicas, algoritmos, e sistemas computacionais (softwares e hardwares) que são desenvolvidos para: a) reduzir o tempo de execução dos programas para um determinado volume de dados e/ou b) aumentar o volume de dados mantendo o tempo de execução dos programas dentro de limites aceitáveis para o problema que se quer resolver. Dentro desse contexto, uma das atividades chaves corresponde a analisar e entender o comportamento de algoritmos e sistemas computacionais para que se possa atingir o máximo desempenho possível para um determinado software executando numa plataforma computacional específica.

Um dos grandes problemas que permeiam não só a área da Computação de Alto Desempenho mas também a área da computação como um todo são as etapas de transferência de dados. No caso específico de sistemas baseados na arquitetura de computação paralela Compute Unified Device Architecture (CUDA), da NVIDIA, a transferência de dados entre a memória da Unidade Central de Processamento - *Central processing unit* (CPU) e a memória da Unidades de Processamento Gráfico - *Graphics Processing Unit* (GPU) é um potencial ponto de perda de desempenho do código.

Uma das possíveis formas de eliminar este problema é fazendo overlapping entre o cômputo da GPU e a transferência de dados da GPU/CPU e CPU/GPU através de comunicação assíncrona [KIRK 2010]. Por isso, é importante o estudo de métodos de sincronismo e assincronismo em comunicação de dados entre CPU e GPU na arquitetura CUDA. Este trabalho objetiva-se a avaliar métodos que podem ser utilizados para reduzir ou eliminar a influência do tempo de comunicação da CPU com o dispositivo CUDA sobre o tempo total de execução de uma aplicação paralela.

Para tal, no entanto, é necessário que haja um estudo de caso adequado. Os seres humanos tornaram-se capazes de colocar satélites em órbita para os mais diversos fins. No entanto, tais equipamentos possuem uma vida útil. Em outras palavras, após um tempo determinado, passam a parar de funcionar adequadamente.

Um dos grandes problemas da nova era tecnológica encontra-se justamente nos detritos que são gerados por estes equipamentos; o chamado “lixo espacial”. Segundo dados da NASA, mais de 500 mil objetos singulares caracterizados como “lixo espacial” orbitam atualmente o planeta, numa velocidade rápida o bastante para danificar os caríssimos equipamentos em órbita e estações espaciais.

Diversas possíveis soluções existem neste campo. Uma delas consiste na seguinte proposta: Um equipamento capaz de encontrar tais detritos, alinhar-se à eles e atirá-los de volta à superfície da terra. Para que tal equipamento funcione, são necessários diversos cálculos físicos de alta demanda de cômputo, na tentativa de solucionar o problema caricaturizado como o problema "Rendezvous".

O trabalho aqui apresentado descreve também o processo de elaboração do caso de uso que envolve o código do Rendezvous, projeto da área da astronomia, que busca apresentar uma solução para descobrir as configurações físicas com as quais um veículo espacial deve ser criado para que o mesmo seja capaz de alcançar detritos espaciais e executar a ação corretiva para a qual foi programado.

## MATERIAIS E MÉTODOS

Para a realização deste trabalho foi utilizada a placa de vídeo Tesla C2070 (Tesla, 2011) baseada na arquitetura CUDA que possui 448 cores e 6 GB de memória. Esta placa foi devidamente instalada e configurada em um servidor com Sistema Operacional (SO) CentOS 6.7, processador Pentium 4, 320 GB de HD e 4 GB de memória RAM.

A primeira fase do trabalho baseou-se em pesquisar as técnicas atualmente utilizadas para otimização de memória na arquitetura CUDA ao passo que esta é uma das mais importantes áreas para performance nesse tipo de tecnologia. Para diminuir o tempo gasto com transferência de dados foram analisados duas abordagens principais: a) o Assincronismo entre transferência de memória e a execução de tarefas a partir da CPU e b) a cópia concorrente utilizando múltiplas Streams (fluxos).

Para avaliar as abordagens encontrada foram inicialmente utilizados casos de uso disponibilizados nos exemplos do CUDA SDK (Software Development Kit) com pequenas alterações: a) assincronismo - para verificar a presença de sobreposição - *overlapping* - de execução em CPU e execução em *kernel* GPU; e b) *Simple_Multi_Copy_and_Compute* - que ilustra o uso de *CUDA Streams* para obter a sobreposição da execução de *Kernels* com cópias entre *Host* e *Device*.

Com o caso de uso para verificar a aplicação real da metodologia desenvolvida em vista e fazendo o uso da linguagem de programação C, foi dado início ao desenvolvimento do código que modela a equação do Rendezvous de forma serial. O problema envolve um algoritmo de força bruta com o intuito de encontrar qual o conjunto de três elementos, chamados de "Gama", "Chi" e "Velocidade de Exaustão (ve)", responsáveis por aproximar o resultado da equação de uma igualdade, permitindo, assim, encontrar o "Rendezvous" (ou ponto de encontro).

Para avaliar se os dados resultantes da execução condiziam com a resolução do Rendezvous na forma análitica, foi desenvolvida uma planilha na qual, dado o conjunto de variáveis de entrada e as três variáveis físicas a serem encontradas, é possível analisar passo a passo o valor que cada um dos coeficientes que envolvem o Rendezvous assume.

Em seguida foi desenvolvido o código paralelo utilizando a plataforma de computação paralela CUDA e fazendo uso da metodologia de comunicação assíncrona.

Por fim foram documentados todos os códigos desenvolvidos para o caso de uso envolvendo o Rendezvous bem como organizado todos os exemplos de arquivos de entrada e saída, além da documentação sobre a forma de compilação e execução dos códigos.

## RESULTADOS E DISCUSSÃO

Como resultado do trabalho pôde-se estabelecer uma metodologia assincronismo e Overlapping de funções executadas em GPU a qual pode ser executada acompanhando os passos apresentados no tutorial. O tutorial desenvolvido por este projeto pode ser acessados na *homepage* do Laboratório de Computação de Alto Desempenho da UEFS (LACAD, 2016).

O algoritmo desenvolvido utilizando a arquitetura CUDA fez uso dos índices dos blocos e threads definidos a partir da chamada de execução da função executada na GPU (kernel). Cada uma das *threads* criadas tinha a responsabilidade de executar todas as operações matemática que envolviam o valor do rendezvous e descobrir o resultado da equação para o conjunto de dados de entrada e o conjunto de variáveis físicas definidas pelos índices que rotulavam a *thread*.

Na Figura 1 a seguir podemos ver o perfil da aplicação do Rendezvous utilizando a ferramenta de medição de desempenho NVPROF (Nvprof, 2016), na qual podemos notar que 91.85% do tempo gasto na execução da aplicação é com a instrução e cópia de memória do Device para o Host.

```
==22770== NVPROF is profiling process 22770, command: ./a.out v-500-eentradas.dat
==22770== Profiling application: ./a.out v-500-eentradas.dat
==22770== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 91.85%  21.0585s  11632321  1.8100us  1.7280us  2.9440us  [CUDA memcpy DtoH]
  7.94%  1.82061s        69  26.386ms  26.360ms  26.411ms  calcularRendezvousDevice(double*)
  0.20%  46.952ms       137  342.71us  332.76us  344.44us  [CUDA memset]
  0.00%  329.43us       345     954ns     896ns  1.3760us  [CUDA memcpy HtoD]
^C
```

**Figura 1:** Perfil da aplicação do Rendezvous Cuda usando NVPROF

O código do Rendezvous para a arquitetura CUDA desenvolvido pode ser visualizado no repositório do Rendezvous (Rendezvous 2017). Todo código foi documentado e teve seu manual de compilação e execução desenvolvido para que atuais e futuros membros do Laboratório de Computação de Alto Desempenho da UEFS sejam capazes de fazer manutenção no código.

A partir dos códigos desenvolvidos foi possível fazer medições de tempo comparando as execuções em série utilizando a linguagem de programação C, em paralelo usando a arquitetura CUDA e em paralelo usando a arquitetura CUDA em conjunto com a metodologia de assincronismo entre CPU e GPU estudada. Na Tabela 1 a seguir pode-se visualizar uma comparação entre o tempo de execução desses 3 códigos:

Tabela 1. Comparativo de tempo dos programas desenvolvidos: Serial, Paralelo em CUDA, Paralelo em CUDA assíncrono

| | **real** | **user** | **sys** |
|---|---|---|---|
| **Serial** | 53m42s | 52m35s | 0m23s |

| | | | |
|---|---|---|---|
| **Paralelo** | 44m26s | 28m27s | 4m17s |
| **Paralelo Assíncrono** | 33m28s | 27m50s | 3m36s |

## CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo o estabelecimento de uma metodologia de aplicação de comunicação assíncrona de funções em códigos desenvolvidos para arquitetura CUDA no Laboratório de Computação de Alto Desempenho (LaCAD).

Para isso foram analisadas duas abordagens principais sendo sendo a primeira delas a utilização de de funções nativas da arquitetura para permitir o assincronismo entre funções executadas na GPU e funções executadas na CPU, ou seja, permitir que ambos os códigos de CPU e GPU sejam executados ao mesmo tempo. Enquanto a outra abordagem foi a de utilização de diferentes *Streams* (fluxos de operação) em CUDA

Como parte do desenvolvimento deste trabalho foi produzido um tutorial que permite que o usuário manipule sua aplicação e consiga um ganho de desempenho através do uso da metodologia desenvolvida em conjunto podendo por sua vez analisar o ganho em questão utilizando ferramentas de medição de desempenho que se encontram disponíveis no laboratório

A partir dos estudos e testes feitos em cima das abordagens utilizadas, foi capaz de utilizar a metodologia para desenvolver um código para uma aplicação real dentro do laboratório, código esta que envolveu todo um processo de documentação, descrição, tutoriais de uso e comparações com abordagens seriais do mesmo.

A partir dos estudos das abordagens possíveis foi capaz de concluir que quando deparado com um problema em que se almeja executar código na CPU e na GPU de forma concomitante deve-se analisar a possibilidade de utilizar a função *cudaMemcpyAsync()* e sincronizar quando necessário utilizando a criação e manipulação de eventos através das funções: *cudaEventCreate*(), *cudaEventRecord()* e *cudaEventSynchronize()*.

Já em casos onde o problema esteja em execução de diversas funções kernel que são independentes entre si (Não exista dependência de dados) a abordagem mais recomendada é a de criação de fluxos (*Streams*) diferentes para cada uma das funções kernel.

## REFERÊNCIAS


[KIRK 2010] KIRK, David B. Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann (2010)

NVPROF. 2017 [online] *Cuda Toolkit Documentation - Nvprof.* Homepage: http://docs.nvidia.com/cuda/profiler-users-guide/#nvprof-overview

LaCAD 2017[online] *Laboratório de Computação de Alto Desempenho.* Homepage: http://lacad.uefs.br/wp-content/uploads/2017/08/async.pdf

Rendezvous 2017 [online] Repositório Rendezvous. Homepage: https://github.com/ssscassio/Rendezvous